MySQL Shell 1.0

Abstract

MySQL Shell, is an advanced client and code editor for MySQL Server. This document describes the core features of MySQL Shell. In addition to the provided SQL functionality, similar to `mysql`, MySQL Shell provides scripting capabilities for JavaScript and Python and includes APIs for working with MySQL. X DevAPI enables you to work with both relational and document data, see Using MySQL as a Document Store. AdminAPI enables you to work with InnoDB Cluster, see MySQL AdminAPI.

MySQL Shell 8.0 is highly recommended for use with MySQL Server 8.0 and 5.7. Please upgrade to MySQL Shell 8.0. If you have not yet installed MySQL Shell, download it from the download site.

For notes detailing the changes in each release, see the MySQL Shell Release Notes.

For help with using MySQL, please visit the MySQL Forums, where you can discuss your issues with other MySQL users.

Licensing information.   This product may include third-party software, used under license. If you are using a *Commercial* release of MySQL Shell, see MySQL Shell Commercial License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Commercial release. If you are using a *Community* release of MySQL Shell, see MySQL Shell Community License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Community release.

Document generated on: 2023-05-03 (revision: 75543)

Table of Contents

Chapter 1 MySQL Shell Features

The following features are available in MySQL Shell.

Interactive Code Execution

MySQL Shell provides an interactive code execution mode, where you type code at the MySQL Shell prompt and each entered statement is processed, with the result of the processing printed onscreen.

Supported Languages

MySQL Shell processes code in the following languages: JavaScript, Python and SQL. Any entered code is processed as one of these languages, based on the language that is currently active. There are also specific MySQL Shell commands, prefixed with \, which enable you to configure MySQL Shell regardless of the currently selected language. For more information see Section 4.1, "MySQL Shell Commands".

Batch Code Execution

In addition to the interactive execution of code, MySQL Shell can also take code from different sources and process it. This method of processing code in a noninteractive way is called *Batch Execution*.

As batch execution mode is intended for script processing of a single language, it is limited to having minimal non-formatted output and disabling the execution of commands. To avoid these limitations, use the `--interactive` command-line option, which tells MySQL Shell to execute the input as if it were an interactive session. In this mode the input is processed *line by line* just as if each line were typed in an interactive session. For more information see Section 3.5, "Batch Mode Made Interactive".

Output Formats

MySQL Shell provides output in different formats depending on how it is used: Tabbed, Table and JSON. For more information see Section 3.3, "Output Formats".

Multiple-line Support

Multiple-line code can be written using a command, enabling MySQL Shell to cache multiple lines and then execute them as a single statement. For more information see Section 3.5.1, "Multiple-line Support".

Application Log

MySQL Shell can be configured to log information about the execution process. For more information see Chapter 5, *MySQL Shell Application Log*.

Supported APIs

MySQL Shell includes the following APIs implemented in JavaScript and Python which you can use to develop code that interacts with MySQL.

- The X DevAPI enables you to work with both relational and document data when MySQL Shell is connected to a MySQL server using the X Protocol. For more information, see Using MySQL as a Document Store. For documentation on the concepts and usage of X DevAPI, see X DevAPI User Guide.

- The AdminAPI enables you to work with InnoDB Cluster, which provides an integrated solution for high availability and scalability using InnoDB based MySQL databases, without requiring advanced MySQL expertise. See MySQL AdminAPI.

For specific documentation on the implementation of the APIs see JavaScript and Python.

X Protocol Support

MySQL Shell is designed to provide an integrated command-line client for all MySQL products which support X Protocol. The development features of MySQL Shell are designed for sessions using the X Protocol. MySQL Shell can also connect to MySQL Servers that do not support the X Protocol using the legacy MySQL Protocol. A minimal set of features from the X DevAPI are available for sessions created using the classic MySQL protocol.

Global Session

Interaction with a MySQL Server is done through a Session object. For Python and JavaScript, a Session can be created through the `getSession` and `getNodeSession` functions of the mysqlx module. If a session is created in JavaScript mode using any of these methods, it is available only in JavaScript mode. The same happens if the session is created in Python mode. None of these sessions can be used in SQL mode.

For SQL Mode, the concept of Global Session is supported by the MySQL Shell. A Global Session is created when the connection information is passed to MySQL Shell using command-line options, or by using the `\connect` command.

The Global Session is used to execute statements in SQL mode and the same session is available in both Python or JavaScript modes. When a Global Session is created, a variable called `session` is set in the scripting languages, so you can execute code in the different languages by switching the active mode.

For more information, see Section 2.2, "MySQL Shell Sessions".

Chapter 2 Getting Started with MySQL Shell

Table of Contents

This section describes how to get started with MySQL Shell, explaining how to connect to a MySQL server instance, and how to choose a session type.

2.1 MySQL Shell Connections

MySQL Shell can connect to MySQL Server using both the X Protocol and the classic MySQL protocol. The address of the MySQL Server which you want to connect to can be specified using individual parameters, such as user, hostname and port, or using a Uniform Resource Identifier (URI) type string. The following sections describe these connection methods. See Connecting to the MySQL Server Using Command Options for more background information.

You can configure the MySQL server instance that MySQL Shell is connected to in the following ways:

- When you start MySQL Shell using the command parameters. See Section 2.1.2, "Connecting using Individual Parameters".

- When MySQL Shell is running using the `\connect` command. See Section 4.1, "MySQL Shell Commands".

- When running Python or Java code using the `shell.connect('instance')` method. See JavaScript and Python.

These methods all support Section 2.1.1, "Connecting using a URI String".

> ⚠️ **Important**
>
> Regardless of the method you choose to connect it is important to understand how passwords are handled by MySQL Shell. By default connections are assumed to require a password. The password is requested at the login prompt. To specify a passwordless account use the `--password` option and do not specify a password, or use a `:` after the `user` in a URI type string and do not specify a password.

If you do not specify parameters for a connection the following defaults are used:

- user defaults to the current system user name

- host defaults to localhost

- port defaults to the X Plugin port 33060 when using an X Protocol session, and port 3306 when using a Classic session

MySQL Shell connections using X Protocol *always* use TCP, using Unix sockets is not supported. MySQL Shell connections using MySQL Protocol default to using Unix sockets when the following conditions are met:

- A TCP port is not specified

- A host name is not specified or it is equal to `localhost`

- A socket is provided with a path to a socket file

- A classic session is specified

If a host name is specified but it is not `localhost`, a TCP connection is established. In this case, if a TCP port is not specified the default value of 3306 is used. If the conditions are met for a socket connection but a path to a socket file is not specified then the default socket is used. See Connecting to the MySQL Server Using Command Options.

2.1.1 Connecting using a URI String

You can configure the MySQL Server which MySQL Shell connects to by passing the connection data in URI type string format. Such strings can be used with the `--uri` command option, the MySQL Shell `\connect` command, and the `shell.connect()` method.

The URI type string should use the following format:

```
scheme://[user[:[password]]@]target[:port][/schema][?attribute1=value1&attribute2=value2...
```

> **Important**
>
> Percent encoding must be used for reserved characters in the elements of the URI type string. For example, if you specify a password that includes the `@` character, the character must be replaced by `%40`.

The elements of a URI type string for a MySQL Shell connection are:

- `scheme`: this element is required and specifies the connection protocol to use, currently either `mysql` for classic MySQL protocol and `mysqlx` for X Protocol.

- `user`: this element is optional and specifies the MySQL user account to be used for the authentication process.

- `password`: this element is optional and specifies the password to be used for the authentication process.

> **Warning**
>
> Storing the password in the URI type string is insecure and not recommended.

- `target`: this element is required and specifies the server instance the connection refers to. Can be either TCP connection information, a Unix socket path or a Windows named-pipe. If not specified, *localhost* is used by default.

  - TCP connection information can be either a host name, an IPv4 address, or an IPv6 address. Can include an optional port number in the format `host:port`, where `port` specifies a network port which the target MySQL server is listening on for connections. If not specified, 33060 is used by default for X Protocol connections, and 3306 is the default for classic MySQL protocol connections.

- Unix socket and Windows named-pipe values are local file paths. There are two ways to specify such paths, using percent encoding or surrounding the path with parentheses, removing the need to percent encode characters such as the common directory separator `/`. For example, to connect as `root@localhost` using the Unix socket `/tmp/mysqld.sock` either specify the path using parenthesis as `root@localhost?socket=(/tmp/mysqld.sock)` or using percent encoding as `root@localhost?socket=%2Ftmp%2Fmysqld.sock`.

- `schema`: this element is optional and specifies the database to be set as default when the connection is established.

- `?attribute=value`: this element is optional and specifies a data dictionary that contains options.

If no password is specified using the URI type string, which is recommended, then the password is prompted for. The following examples show how to specify URI type strings with the user name `user`, in each case the password is prompted for:

- An X Protocol connection to a local server instance listening at port 33065.

```
mysqlx://user@localhost:33065
```

- A classic MySQL protocol connection to a local server instance listening at port 3333.

```
mysql://user@localhost:3333
```

- An X Protocol connection to a remote server instance, using a host name, an IPv4 address and an IPv6 address.

```
mysqlx://user@server.example.com/
mysqlx://user@198.51.100.14:123
mysqlx://user@[2001:db8:85a3:8d3:1319:8a2e:370:7348]
```

- An optional path can be specified, which represents a database schema.

```
mysqlx://user@198.51.100.1/world%5Fx
mysqlx://user@198.51.100.2:33060/world
```

- An optional query can be specified, consisting of values in the form of a `key=value` pair or as a single `key`. The `,` character is used as a separator for values, a combination of multiple pairs and keys can be specified. Values can be of type list, list values are ordered by appearance. Strings must be percent encoded.

```
ssluser@127.0.0.1?ssl-ca%3D%2Froot%2Fclientcert%2Fca-cert.pem%26ssl-cert%3D%2Fro\
ot%2Fclientcert%2Fclient-cert.pem%26ssl-key%3D%2Froot%2Fclientcert%2Fclient-key
.pem
```

Although using a passwordless account is insecure and not recommended, you can specify a user without a password using a `:` after the user name, for example:

```
mysqlx://user:@localhost
```

2.1.2 Connecting using Individual Parameters

In addition to specifying connection parameters using a URI type string, it is also possible to define the connection data when starting MySQL Shell using separate command parameters for each value. For a full reference of MySQL Shell command options see Section A.1, "mysqlsh — The MySQL Shell".

Use the following connection related parameters:

- `--dbuser` (`-u`) *value*

- `--dbpassword` *value*

- `--host` (`-h`) *value*

- `--port` (`-P`) *value*

- `--schema` (`-D`) *value*

- `--password` (`-p`)

- `--socket` (`-S`)

The first 5 parameters match the elements used in the URI type string format described at Section 2.1.1, "Connecting using a URI String".

The `--password` parameter indicates the user should connect *without* a password.

For consistency, the following aliases are supported for some parameters:

- `--user` is equivalent to `--dbuser`

- `--password` is equivalent to `--dbpassword`

- `--database` is equivalent to `--schema`

When parameters are specified in multiple ways, for example using both the `--uri` option and specifying individual parameters such as `--user`, the following rules apply:

- If an argument is specified more than once the value of the last appearance is used.

- If both individual connection arguments and `--uri` are specified, the value of `--uri` is taken as the base and the values of the individual arguments override the specific component from the base URI.

  For example to override *user* from the URI:

  ```
  $> mysqlsh --uri user@localhost:33065 --user otheruser
  ```

The following examples show how to use command parameters to specify connections. Attempt to establish an X Protocol connection with a specified user at port 33065.

```
$> mysqlsh --mysqx1 -u user -h localhost -P 33065
```

Attempt to establish a classic MySQL protocol connection with a specified user.

```
$> mysqlsh --mysql -u user -h localhost
```

2.1.3 Using Encrypted Connections

Using encrypted connections is possible when connecting to a TLS (sometimes referred to as SSL) enabled MySQL server. Much of the configuration of MySQL Shell is based on the options used by MySQL server, see Using Encrypted Connections for more information.

To configure an encrypted connection at startup of MySQL Shell, use the following command options:

- `--ssl` : Deprecated, to be removed in a future version. This option enables or disables encrypted connections.

- `--ssl-mode` : This option specifies the security state of the connection to the server.

- `--ssl-ca=`*`filename`*: The path to a file in PEM format that contains a list of trusted SSL Certificate Authorities.

- `--ssl-capath=`*`directory`*: The path to a directory that contains trusted SSL Certificate Authority certificates in PEM format.

- `--ssl-cert=`*`filename`*: The name of the SSL certificate file in PEM format to use for establishing an encrypted connection.

- `--ssl-cipher=`*`name`*: The name of the SSL cipher to use for establishing an encrypted connection.

- `--ssl-key=`*`filename`*: The name of the SSL key file in PEM format to use for establishing an encrypted connection.

- `--ssl-crl=`*`name`*: The path to a file containing certificate revocation lists in PEM format.

- `--ssl-crlpath=`*`directory`*: The path to a directory that contains files containing certificate revocation lists in PEM format.

- `--tls-version=`*`version`*: The TLS protocols permitted for encrypted connections.

Alternatively the SSL options can be encoded as part of a URI type string as part of the query element. The available SSL options are the same as those listed above, but written without the proceeding hyphens. For example `ssl-ca` is the equivalent of `--ssl-ca`.

Paths specified in a URI type string must be percent encoded, for example:

```
ssluser@127.0.0.1?ssl-ca%3D%2Froot%2Fclientcert%2Fca-cert.pem%26ssl-cert%3D%2Fro\
ot%2Fclientcert%2Fclient-cert.pem%26ssl-key%3D%2Froot%2Fclientcert%2Fclient-key
.pem
```

See Section 2.1.1, "Connecting using a URI String" for more information.

2.1.4 Connections in JavaScript and Python

When a connection is made using the command options or by using any of the MySQL Shell commands, a global session object is created. This session is global because once created, it can be used in any of the MySQL Shell execution modes.

Any global session object is available in JavaScript or Python modes because a variable called **session** holds a reference to it.

In addition to the global session object, sessions can be established and assigned to a different variable by using the functions available in the `mysql` and `mysqlx` JavaScript and Python modules.

For example, the following functions are provided by these modules:

- `mysqlx.getSession(connectionData[, `*`password`*`])`

  The returned object can be Session if the object was created or retrieved using a Session instance, and ClassicSession if the object was created or retrieved using a ClassicSession instance.

- `mysql.getClassicSession(connectionData[, `*`password`*`])`

  The returned object is a ClassicSession which uses classic MySQL protocol and has a limited development API.

`connectionData` can be either a URI type string as specified at Section 2.1.1, "Connecting using a URI String" or a dictionary containing the connection parameters.

The following example shows how to create a Session using the X Protocol:

```
mysql-js> var mysession1=mysqlx.getSession('root@localhost:33060', 'password');
mysql-js> session
<Session:root@localhost>
mysql-js>
```

The following example shows how to create a ClassicSession:

```
mysql-js> var mysession2=mysql.getClassicSession('root@localhost:3306', 'password');
mysql-js> session
<ClassicSession:root@localhost:3306>
mysql-js>
```

2.1.4.1 Using Encrypted Connections in Code

To establish an encrypted connection, set the SSL information in the `connectionData` dictionary. For example:

```
mysql-js> var session=mysqlx.getSession({host: 'localhost',
user: 'root',
password: 'password',
ssl_ca: "path_to_ca_file",
ssl_cert: "path_to_cert_file",
ssl_key: "path_to_key_file"});
```

2.2 MySQL Shell Sessions

This section explains the different types of sessions in MySQL Shell and how to create and configure them.

2.2.1 MySQL Shell Sessions Explained

MySQL Shell is a unified interface to operate MySQL Server through scripting languages such as JavaScript or Python. To maintain compatibility with previous versions, SQL can also be executed in certain modes. A connection to a MySQL server is required. In MySQL Shell these connections are handled by a *Session* object.

The following types of Session object are available:

- *NodeSession*: Use this session type for new application development to communicate with MySQL server instances which have the X Protocol enabled. It offers the best integration with MySQL Server, and therefore, it is used by default.

- *ClassicSession* Use this session type to interact with MySQL Servers that do not have the X Protocol enabled. The development API available for this type of session is very limited. For example, there are no CRUD operations, no collection handling, and binding is not supported.

> ⚠️ **Important**
>
> `ClassicSession` is specific to MySQL Shell and cannot be used with other implementations of X DevAPI, such as MySQL Connectors.

Choosing a MySQL Shell Session Type

MySQL Shell creates a Session object by default. You can either configure the session type using MySQL Shell command options, the `scheme` element of a URI type string, or provide an option to the `\connect`

command. To choose which type of session should be created when starting MySQL Shell, use one of these options:

- `--sqln` creates a Node session, connected using X Protocol.

- `--sqlc` creates a ClassicSession, connected using classic MySQL protocol.

To choose which type of session to use when defining a URI type string use one of these options:

- Specify `mysqlx` to create an X Protocol session. The X Plugin must be installed on the server instance, see Using MySQL as a Document Store for more information.

- Specify `mysql` to create a classic MySQL protocol session.

For more information, see Section 2.1.1, "Connecting using a URI String".

Creating a Session Using Shell Commands

If you open MySQL Shell without specifying connection parameters, MySQL Shell opens without an established global session. It is possible to establish a global session once MySQL Shell has been started using the MySQL Shell `\connect URI` command, where `URI` is a URI type string as defined at Section 2.1.1, "Connecting using a URI String". For example:

- `\connect URI`: Creates a Node session. Attempts to use X Protocol by default, and falls back to classic MySQL protocol.

- `\connect -n URI`: Creates a Node session.

- `\connect -c URI`: Creates a ClassicSession using classic MySQL protocol.

For example:

```
mysql-js> \connect mysqlx://user@localhost:33060
```

Alternatively, use the `shell.connect('URI'` method. For example this is equivalent to the above `\connect>` command:

```
mysql-js> shell.connect('mysqlx://user@localhost:33060')
```

2.3 MySQL Shell Global Variables

MySQL Shell reserves certain variables as global variables, which are assigned to commonly used objects in scripting. This section describes the available global variables and provides examples of working with them. The global variables are:

- `session` represents the global session if one has been established.

- `db` represents a schema if one has been defined, for example by a URI type string.

- `dba` represents the AdminAPI, a component of InnoDB Cluster which enables you to administer clusters of server instances. See MySQL AdminAPI.

- `shell` provides general purpose functions, for example to configure MySQL Shell.

Important

These words are reserved and cannot be used, for example as names of variables.

By using these global objects, MySQL Shell provides interactive error resolution for common situations. For example:

- Attempting to use an undefined `session` global variable.

- Attempting to retrieve an nonexistent schema using `session`.

- Attempting to use an undefined `db` global variable.

Undefined Global Session

The global `session` variable is set when a global session is established. When a global session is established, issuing a `session` statement in MySQL Shell displays the session type and its URI as follows:

```
mysql-js> session
<NodeSession:root@localhost:33060>
mysql-js>
```

If no global session has been established, MySQL Shell displays the following:

```
mysql-js> session
<Undefined>
mysql-js>
```

If you attempt to use the `session` variable when no global session is established, interactive error resolution starts and you are prompted to provide the required information to establish a global session. If the session is successfully established, it is assigned to the `session` variable. The prompts are:

- An initial prompt explains that no global session is established and asks if one should be established.

- If you choose to set a global session, the session type is requested.

- The URI type string to connect to the server instance is requested. See Section 2.1.1, "Connecting using a URI String".

- If required, a password is requested.

For example:

```
mysql-js> session.uri
The global session is not set, do you want to establish a session?

1) MySQL Document Store Session through X Protocol
2) Classic MySQL Session

Please select the session type or ENTER to cancel: 2
Please specify the MySQL server URI: root@localhost
Enter password:
root@localhost:
mysql-js> session
<ClassicSession:root@localhost:>
```

Undefined db Variable

The global `db` variable is set when a global session is established and a default schema is configured. For example, using a URI type string such as `root@localhost/sakila` to establish a global session

connected to the MySQL Server at `localhost`, on port 33060, as the user `root`, assigns the schema `sakila` to the global variable `db`. Once a schema is defined, issuing `db` at the MySQL Shell prompt prints the schema name as follows:

```
mysql-js> db
<Schema:world_x>
mysql-js>
```

If there is no global session established, the following is displayed:

```
mysql-js> db
<Undefined>
mysql-js>
```

If you attempt to use the `db` variable when no global session has been established, the following error is displayed:

```
mysql-js> db.getCollections()
LogicError: The db variable is not set, establish a global session first.
at (shell):1:2
in db.getCollections()
^
```

If a global session has been established but you attempt to use an undefined `db`, interactive error resolution begins and you are prompted to define an active schema by providing the schema name. If this succeeds the `db` variable is set to the defined schema. For example:

```
mysql-js> db.getCollections()
The db variable is not set, do you want to set the active schema? [y/N]:y
Please specify the schema:world_x
[
<Collection:countryinfo>
]
mysql-js> db
<Schema:world_x>
mysql-js>
```

Retrieving an Nonexistent Schema

If you attempt to use `session` to retrieve an nonexistent schema, interactive error resolution provides the option to create the schema.

```
mysql-js> var mySchema = session.getSchema('my_test')
The schema my_test does not exist, do you want to create it? [y/N]: y

mysql-js> mySchema
<Schema:my_test>
mysql-js>
```

In all cases, if you do not provide the information required to resolve each situation, a proper result of executing the requested statement on an undefined variable is displayed.

Chapter 3 MySQL Shell Code Execution

Table of Contents

This section explains how code execution works in MySQL Shell.

3.1 Interactive Code Execution

The default mode of MySQL Shell provides interactive execution of database operations that you type at the command prompt. These operations can be written in JavaScript, Python or SQL depending on the curent Section 3.4, "Active Language". When executed, the results of the operation are displayed on-screen.

As with any other language interpreter, MySQL Shell is very strict regarding syntax. For example, the following JavaScript snippet reads and prints the documents in a collection:

```
var mysqlx = require('mysqlx').mysqlx;
var mySession = mysqlx.getSession('user:pwd@localhost');
var result = mySession.world_x.countryinfo.find().execute();
var record = result.fetchOne();
while(record){
print(record);
record = result.fetchOne();
}
```

As seen above, the call to `find()` is followed by the `execute()` function. CRUD database commands are only actually executed on the MySQL Server when `execute()` is called. However, when working with MySQL Shell interactively, `execute()` is implicitly called whenever you press `Return` on a statement. Then the results of the operation are fetched and displayed on-screen. The rules for when you need to call `execute()` or not are as follows:

- When using MySQL Shell in this way, calling `execute()` becomes optional on:

  - `Collection.add()`

  - `Collection.find()`

  - `Collection.remove()`

  - `Collection.modify()`

  - `Table.insert()`

  - `Table.select()`

- `Table.delete()`

- `Table.update()`

- `NodeSession.sql()`

- Automatic execution is disabled if the object is assigned to a variable. In such a case calling `execute()` is mandatory to perform the operation.

- When a line is processed and the function returns any of the available `Result` objects, the information contained in the Result object is automatically displayed on screen. The functions that return a Result object include:

  - The SQL execution and CRUD operations (listed above)

  - Transaction handling and drop functions of the session objects in both mysql and mysqlx modules: -

    - `startTransaction()`

    - `commit()`

    - `rollback()`

    - `dropSchema()`

    - `dropTable()`

    - `dropCollection()`

    - `dropView()`

    - `ClassicSession.runSql()`

Based on the above rules, the statements needed in the MySQL Shell in interactive mode to establish a session, query, and print the documents in a collection are:

```
mysql-js> var mysqlx = require('mysqlx').mysqlx;
mysql-js> var mySession = mysqlx.getSession('user:pwd@localhost');
```

No call to `execute()` is needed and the Result object is automatically printed.

```
mysql-js> mySession.world_x.countryinfo.find();
```

Formatting Results Vertically

When executing SQL using MySQL Shell you can display results in a column-per-row format with the `\G` command, in a similar way to `mysql`. If a statement is terminated with `\G` instead of the active delimiter (which defaults to `;`), it is executed by the server and the results are displayed in vertical format, regardless of the current default output format. For example issuing a statement such as

```
SELECT * FROM mysql.user \G
```

displays the results vertically.

Multiple SQL statements on the same line which are separated by `\G` are executed separately as if they appeared one per line., for example

```
select 1\Gselect 2\Gselect 3\G
```

In other words `\G` functions as a normal delimiter.

3.2 Batch Code Execution

As well as interactive code execution, MySQL Shell provides batch code execution from:

- A file loaded for processing.

- A file containing code that is redirected to the standard input for execution.

- Code from a different source that is redirected to the standard input for execution.

The input is processed based on the current programming language selected in MySQL Shell, which defaults to JavaScript. For example:

Loading JavaScript code from a file for batch processing.

```
$> mysqlsh --file code.js
```

Redirecting a JavaScript file to standard input for execution.

```
$> mysqlsh < code.js
```

Redirecting SQL to standard input for execution.

```
$> echo "show databases;" | mysqlsh --sql --uri root@198.51.100.141:33060
```

Executable Scripts

Starting with version 1.0.4, on Linux you can create executable scripts that run with MySQL Shell by including a `#!` line as the first line of the script. This line should provide the full path to MySQL Shell and include the `--file` option. For example:

```
#!/usr/local/mysql-shell/bin/mysqlsh --file
print("Hello World\n");
```

The script file must be marked as executable in the filesystem. Running the script invokes MySQL Shell and it executes the contents of the script.

3.3 Output Formats

The output of the commands processed on the server can be formatted in different ways. This section details the different available output formats.

3.3.1 Table Format

The table format is used by default when MySQL Shell is in interactive mode. The output is presented as a formatted table for a better view and to aid analysis.

```
mysql-sql> select * from sakila.actor limit 3;
+----------+-------------+---------------+---------------------+
| actor_id | first_name  | last_name     | last_update         |
```

```
+----------+------------+---------------+---------------------+
|        1 | PENELOPE   | GUINESS       | 2006-02-15 4:34:33  |
|        2 | NICK       | WAHLBERG      | 2006-02-15 4:34:33  |
|        3 | ED         | CHASE         | 2006-02-15 4:34:33  |
+----------+------------+---------------+---------------------+
3 rows in set (0.00 sec)

mysql-sql>
```

To get this output format when running in batch mode, use the `--table` command-line option.

3.3.2 Tab Separated Format

This format is used by default when running MySQL Shell in batch mode, to have better output for automated analysis.

```
>echo "select * from sakila.actor limit 3;" | mysqlsh --classic --uri root@198.51.100.141:33460
actor_id        first_name      last_name       last_update
1       PENELOPE        GUINESS 2006-02-15 4:34:33
2       NICK    WAHLBERG        2006-02-15 4:34:33
3       ED      CHASE   2006-02-15 4:34:33
```

3.3.3 JSON Format Output

MySQL Shell supports the JSON format for output and it is available both in interactive and batch mode. This output format can be enabled using the `--json` command-line option:

JSON Format in Batch Mode

```
$>echo "select * from sakila.actor limit 3;" | mysqlsh --json --sqlc --uri root@198.51.100.141:3306
{"duration":"0.00 sec","info":"","row_count":3,"rows":[[1,"PENELOPE","GUINESS",{"year":2006,"month":1,"day

$>echo "select * from sakila.actor limit 3;" | mysqlsh --json=raw --sqlc --uri root@198.51.100.141:3306
{"duration":"0.00 sec","info":"","row_count":3,"rows":[[1,"PENELOPE","GUINESS",{"year":2006,"month":1,"day

$>echo "select * from sakila.actor limit 3;" | mysqlsh --json=pretty --sqlc --uri root@198.51.100.141:3306
{
"duration": "0.00 sec",
"info": "",
"row_count": 3,
"rows": [
[
1,
"PENELOPE",
"GUINESS",
{
"year": 2006,
"month": 1,
"day": 15,
"hour": 4,
"minute": 34,
"second": 33.0
}
],
[
2,
"NICK",
"WAHLBERG",
{
"year": 2006,
"month": 1,
"day": 15,
```

```
"hour": 4,
"minute": 34,
"second": 33.0
}
],
[
3,
"ED",
"CHASE",
{
"year": 2006,
"month": 1,
"day": 15,
"hour": 4,
"minute": 34,
"second": 33.0
}
]
],
"warning_count": 0
}
$>
```

JSON Format in Interactive Mode (started with --json=raw)

```
mysql-sql> select * from sakila.actor limit 3;
{"duration":"0.00 sec","info":"","row_count":3,"rows":[[1,"PENELOPE","GUINESS",{"year":2006,"month":1,

mysql-sql>
```

JSON Format in Interactive Mode (started with --json=pretty)

```
mysql-sql> select * from sakila.actor limit 3;
{
"duration": "0.00 sec",
"info": "",
"row_count": 3,
"rows": [
[
1,
"PENELOPE",
"GUINESS",
{
"year": 2006,
"month": 1,
"day": 15,
"hour": 4,
"minute": 34,
"second": 33.0
}
],
[
2,
"NICK",
"WAHLBERG",
{
"year": 2006,
"month": 1,
"day": 15,
"hour": 4,
"minute": 34,
"second": 33.0
}
],
[
```

```
      3,
      "ED",
      "CHASE",
      {
      "year": 2006,
      "month": 1,
      "day": 15,
      "hour": 4,
      "minute": 34,
      "second": 33.0
      }
      ]
      ],
      "warning_count": 0
      }

mysql-sql>
```

3.3.4 Result Metadata

When an operation is executed, in addition to any results returned, some additional information is available. This includes information such as the number of affected rows, warnings, duration, and so on, when any of these conditions is true:

- JSON format is being used for the output

- MySQL Shell is running in interactive mode.

3.4 Active Language

MySQL Shell can execute SQL, JavaScript or Python code, but only one language can be active at a time. The active mode determines how the executed statements are processed:

- If using SQL mode, statements are processed as SQL which means they are sent to the MySQL server for execution.

- If using JavaScript mode, statements are processed as JavaScript code.

- If using Python mode, statements are processed as Python code.

When running MySQL Shell in interactive mode, activate a specific language by entering the commands: `\sql`, `\js`, `\py`.

When running MySQL Shell in batch mode, activate a specific language by passing any of these command-line options: `--js`, `--py` or `--sql`. The default mode if none is specified is JavaScript.

Use MySQL Shell to execute the content of the file `code.sql` as SQL.

```
$> mysqlsh --sql < code.sql
```

Use MySQL Shell to execute the content of the file `code.js` as JavaScript code.

```
$> mysqlsh < code.js
```

Use MySQL Shell to execute the content of the file `code.py` as Python code.

```
$> mysqlsh --py < code.py
```

3.5 Batch Mode Made Interactive

This section describes code execution in batch mode.

- In batch mode, all the command logic described above is not available, only valid code for the active language can be executed.

- When processing SQL code, it is executed statement by statement using the following logic: read/process/print result.

- When processing non-SQL code, it is loaded entirely from the input source and executed as a unit.

Use the `--interactive` (or `-i`) command-line option to configure MySQL Shell to process the input source as if it were being issued in interactive mode; this enables all the features provided by the Interactive mode to be used in batch processing.

> **Note**
>
> In this case, whatever the source is, it is read line by line and processed using the interactive pipeline.

3.5.1 Multiple-line Support

It is possible to specify statements over multiple lines. When in Python or JavaScript mode, multiple-line mode is automatically enabled when a block of statements starts like in function definitions, if/then statements, for loops, and so on. In SQL mode multiple line mode starts when the command \ is issued.

Once multiple-line mode is started, the subsequently entered statements are cached.

For example:

```
mysql-sql> \
... create procedure get_actors()
... begin
...   select first_name from sakila.actor;
... end
...
mysql-sql>
```

Chapter 4 Configuring MySQL Shell

Table of Contents

This section explains how to configure MySQL Shell using commands executable from the interactive code editor and command options. For a description of MySQL Shell command options, see Section A.1, "mysqlsh — The MySQL Shell".

4.1 MySQL Shell Commands

MySQL Shell provides commands which enable you to modify the execution environment of the code editor, for example to configure the active programming language or a MySQL Server connection. The following table lists the commands that are available regardless of the currently selected language. As commands need to be available independent of the *execution mode*, they start with an escape sequence, the \ character.

Command	Alias/Shortcut	Description
\help	\h or \?	Prints help about MySQL Shell commands.
\quit	\q or \exit	Exit MySQL Shell.
\		In SQL mode, begin multiple-line mode. Code is cached and executed when an empty line is entered.
\status	\s	Show the current MySQL Shell status.
\js		Switch execution mode to JavaScript.
\py		Switch execution mode to Python.
\sql		Switch execution mode to SQL.
\connect	\c	Connect to a MySQL Server with a URI using an Node session (X Protocol).
\connect_node	\cn	(Removed in version 1.0.4, use \connect -n) Connect to a MySQL Server with a URI using a Node session.
\connect_classic	\cc	(Removed in version 1.0.4, use \connect -c) Connect to a MySQL Server with a URI using a Classic session (MySQL Protocol).
\use		Specify the schema to use.
\source	\.	Execute a script file using the active language.

Command	Alias/Shortcut	Description
\warnings	\W	Show any warnings generated by a statement.
\nowarnings	\w	Do not show any warnings generated by a statement.
\lsconn	\lsc	Print the connection data for the stored sessions.
\saveconn	\savec	Save connection data of a session, optionaly use -f to force overwriting an existing connection.
\addconn	\addc	(Removed in version 1.0.4, see \saveconn) Store the connection data of a session.
\rmconn		Removes a stored session.
\chconn		(Removed in version 1.0.4, see \saveconn) Updates a stored session.

Help Command

The \help command can be used with or without parameters. When used without parameters a general help is printed including information about:

- Available commands.

- Available commands for the active mode.

When used with a parameter, the parameter must be a valid command. If that is the case, help for that specific command is printed including:

- Description

- Supported aliases if any

- Additional help if any

For example:

```
\help connect
```

If the parameter is not a valid command, the general help is printed.

Connect Command

The \connect command is used to connect to a MySQL Server using an URI type string. See Section 2.1.1, "Connecting using a URI String".

For example:

```
\connect root@localhost:3306
```

If a password is required you are prompted for it.

Use the -n option to create a Node session, using the X Protocol to connect to a single server. For example:

```
\connect -n root@localhost:3306
```

Use the `-c` option to create a Classic session, enabling you to use the MySQL Protocol to issue SQL commands directly on a server. For example:

```
\connect -c root@localhost:3306
```

Status Command

The `\status` command displays information about the current global connection. This includes information about the server connected to, the character set in use, uptime, and so on.

Source Command

The `\source` command is used to execute code from a script at a given path. For example:

```
\source /tmp/mydata.sql
```

You can execute either SQL, JavaScript or Python code. The code in the file is executed using the active language, so to process SQL code the MySQL Shell must be in SQL mode.

> **Warning**
>
> As the code is executed using the active language, executing a script in a different language than the currently selected execution mode language could lead to unexpected results.

Use Command

The `\use` command enables you to choose which schema is active, for example:

```
\use schema_name
```

The `\use` command requires a global development session to be active. The `\use` command sets the current schema to the specified `schema_name` and updates the `db` variable to the object that represents the selected schema.

Chapter 5 MySQL Shell Application Log

This section explains the logging provided by MySQL Shell, where to find logs and how to configure the level of logging.

MySQL Shell can be configured to generate an application log file with information about issues of varying severity. You can use this information to verify the state of MySQL Shell while it is running. The log format is plain text and entries contain a timestamp and description of the problem. For example:

```
2016-04-05 22:23:01: Error: Default Domain: (shell):1:8: MySQLError: You have an error
in your SQL syntax; check the manual that corresponds to your MySQL server version for
the right syntax to use near '' at line 1 (1064) in session.sql("select * from t
limit").execute().all();
```

The amount of information to add to the log can be configured using `--log-level`. See Configuring Logging.

MySQL Shell Log File Location

The location of the log file is the user configuration path and the file is named `mysqlsh.log`.

Log File on Windows

On Windows, the default path to the log file is `%APPDATA%\MySQL\mysqlsh\mysqlsh.log`

To find the location of `%APPDATA%` on your system, echo it from the comand-line. For example:

```
C:>echo %APPDATA%
C:\Users\exampleuser\AppData\Roaming
```

On Windows, the path is determined by the result of gathering the `%APPDATA%` folder specific to that user, and then appending `MySQL\mysqlsh`. Using the above example results in `C:\Users\exampleuser\AppData\Roaming\MySQL\mysqlsh\mysqlsh.log`.

Log File on Unix-based Systems

For a machine running Unix, the default path is `~/.mysqlsh/mysqlsh.log` where "~" represents your home directory. The environment variable `HOME` also represents the home directory. Appending `.mysqlsh` to the this home directory determines the default path to the logs. For example:

```
$>echo $HOME
/home/exampleuser
```

Therefore the location of the MySQL Shell file on this system is `/home/exampleuser/.mysqlsh/mysqlsh.log`.

These paths can be overridden on all platforms by defining the environment variable `MYSQL_USER_CONFIG_PATH`. The value of this variable replaces `%APPDATA%` in Windows or `$HOME` in Unix.

Configuring Logging

By default, logging is disabled in MySQL Shell. To enable logging use the `--log-level` command option when starting MySQL Shell. The value assigned to `--log-level` controls the level of detail in the log. The level of logging can be defined using either numeric levels from 1 to 8, or equivalent named levels as shown in the following table.

Log Level Number	Log Level Name	Meaning
1	none	No logging, the default
2	internal	Internal Error
3	error	Errors are logged
4	warning	Warnings are logged
5	info	Information is logged
6	debug	Debug information is logged
7	debug2	Debug with more information is logged
8	debug3	Debug with full information is logged

The numeric and named levels are equivalent. For example there is no difference in logging when starting MySQL Shell in either of these ways:

```
$>mysqlsh --log-level=4
$>mysqlsh --log-level=warning
```

Chapter 6 Customizing MySQL Shell

Table of Contents

MySQL Shell offers the ability to customize the behavior and code execution environment through startup scripts, which are executed when the application is first run. Using such scripts enables you to:

- Add additional search paths for Python or JavaScript modules.

- Override the default prompt used by the Python and JavaScript modes.

- Define global functions or variables.

- Any other possible initialization through JavaScript or Python.

6.1 Working With Start-Up Scripts

When MySQL Shell enters either into JavaScript or Python mode, it searches for startup scripts to be executed. The startup scripts are JavaScript or Python specific scripts containing the instructions to be executed when the corresponding mode is initialized.

Startup scripts must be named as follows:

- For JavaScript mode: `mysqlshrc.js`

- For Python mode: `mysqlshrc.py`

MySQL Shell searches the following paths for these files (in order of execution).

On Windows:

1. `%PROGRAMDATA%MySQLmysqlshmysqlshrc.[js|py]`

2. `%MYSQLSH_HOME%sharedmysqlshmysqlshrc.[js|py]`

3. `<mysqlsh binary path>mysqlshrc.[js|py]`

4. `%APPDATA%MySQLmysqlshmysqlshrc.[js|py]`

On Linux and macOS:

1. `/etc/mysql/mysqlsh/mysqlshrc.[js|py]`

2. `$MYSQLSH_HOME/shared/mysqlsh/mysqlshrc.[js|py]`

3. `<mysqlsh binary path>/mysqlshrc.[js|py]`

4. `$HOME/.mysqlsh/mysqlshrc.[js|py]`

The environment variable `MYSQLSH_HOME` defines the root folder of a standard setup of MySQL Shell. If `MYSQLSH_HOME` is not defined it is automatically calculated based on the location of the MySQL Shell binary, therefore on many standard setups it is not required to define `MYSQLSH_HOME`.

If `MYSQLSH_HOME` is not defined and the MySQL Shell binary is not in a standard install folder structure, then the path defined in option 3 in the above lists is used. If using a standard install or if `MYSQLSH_HOME` points to a standard install folder structure, then the path defined in option 3 is not used.

> **Warning**
>
> The lists above also define the order of searching the paths, so if something is defined in two different scripts, the script executed later takes precedence.

6.2 Adding Module Search Paths

There are two ways to add additional module search paths:

- Through environment variables

- Through startup scripts

6.2.1 Environment Variables

Python uses the `PYTHONPATH` environment variable to allow extending the search paths for python modules. The value of this variable is a list of paths separated by:

- A colon character in Linux and macOS

- A semicolon character in Windows

To achieve this in JavaScript, MySQL Shell supports defining additional JavaScript module paths using the `MYSQLSH_JS_MODULE_PATH` environment variable. The value of this variable is a list of semicolon separated paths.

6.2.2 Startup Scripts

The addition of module search paths can be achieved for both languages through the corresponding startup script.

For Python modify the `mysqlshrc.py` file and append the required paths into the `sys.path` array.

```
# Import the sys module
import sys

# Append the additional module paths
sys.path.append('~/custom/python')
sys.path.append('~/other/custom/modules')
```

For JavaScript the same task is achieved by adding code into the `mysqlshrc.js` file to append the required paths into the predefined `shell.js_module_paths` array.

```
// Append the additional module paths
shell.js.module_paths[shell.js.module_paths.length] = '~/custom/js';
shell.js.module_paths[shell.js.module_paths.length] = '~/other/custom/modules';
```

6.3 Overriding the Default Prompt

MySQL Shell uses a default prompt for both Python (`mysql-py>`) and JavaScript (`mysql-js>`).

You can customize the language specific prompt using the `shell.custom_prompt()` function. This function must return a string that is used as the prompt. To have a custom prompt when MySQL Shell starts, define this function in a startup script. The following example shows how this functionality can be used.

In Python `shell.custom_prompt()` could be defined as:

```python
# Import the sys module
from time import gmtime, strftime

def my_prompt():
  ret_val = strftime("%H:%M:%S", gmtime())

  if session and session.isOpen():
    data = shell.parseUri(session.getUri())

    ret_val = "%s-%s-%s-py> " % (ret_val, data.dbUser, data.host)
  else:
    ret_val = "%s-disconnected-py> " % ret_val

  return ret_val


shell.custom_prompt = my_prompt
```

In JavaScript `shell.custom_prompt()` could be defined as:

```javascript
shell.custom_prompt = function(){
  var now = new Date();

  var ret_val = now.getHours().toString()+ ":" + now.getMinutes().toString() + ":" + now.getSeconds().toSt:

  if (session && session.isOpen()){
    var data = shell.parseUri(session.getUri());

    ret_val += "-" + data.dbUser + "-" + data.host + "-js> ";
  }
  else
    ret_val += "-disconnected-js> ";

  return ret_val;
}
```

The following example demonstrates using the custom prompt functions defined above in startup script. The prompts show the current system time, and if a session is open the current user and host:

```
Welcome to MySQL Shell 1.0.4 Development Preview

Copyright (c) 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type '\help', '\h' or '\?' for help.

Currently in JavaScript mode. Use \sql to switch to SQL mode and execute queries.
14:34:32-disconnected-js> \py
Switching to Python mode...

19:34:39-disconnected-py> \connect root:@localhost
Creating an X Session to root@localhost:33060
No default schema selected.

19:34:50-root-localhost-py> \js
```

```
Switching to JavaScript mode...
14:34:57-root-localhost-js>
```

Appendix A MySQL Shell Command Reference

Table of Contents

This appendix describes the mysqlsh command.

A.1 mysqlsh — The MySQL Shell

MySQL Shell is an advanced command-line client and code editor for MySQL. In addition to SQL, MySQL Shell also offers scripting capabilities for JavaScript and Python. For information about using MySQL Shell, see MySQL Shell 1.0. When MySQL Shell is connected to the MySQL Server through the X Protocol, the X DevAPI can be used to work with both relational and document data, see Using MySQL as a Document Store. MySQL Shell includes the AdminAPI that enables you to work with InnoDB Cluster, see MySQL AdminAPI.

Many of the options described here are related to connections between MySQL Shell and a MySQL Server instance. See Section 2.1, "MySQL Shell Connections" for more information.

mysqlsh supports the following command-line options.

Table A.1 mysqlsh Options

Option Name	Description
--auth-method	Authentication method to use
--classic	Create a classic MySQL protocol session
--database	The schema to use (alias for --schema)
--dba	Enable X Protocol on connection with MySQL 5.7 server
--dbpassword	Password to use when connecting to server
--dbuser	MySQL user name to use when connecting to server
--execute	Execute the command and quit
--file	File to process in batch mode
--force	Continue in SQL and batch modes even if errors occur
--help	Display help message and exit
--host	Host on which MySQL server instance is located
--interactive	Emulate Interactive mode in batch mode
--js, --javascript	Start in JavaScript mode
--json	Print output in JSON format
--log-level	Specify logging level
--no-wizard, --nw	Disable the interactive wizards
--node	Create a NodeSession
--password	Password to use when connecting to server (alias for --dbpassword)

Option Name	Description
--passwords-from-stdin	Read the password from stdin
--port	TCP/IP port number for connection
--py, --python	Start in Python mode
--recreate-schema	Drop and recreate schema
--schema	The schema to use
--show-warnings	Show warnings after each statement if there are any (in SQL mode)
--socket	Unix socket file or Windows named pipe to use (classic MySQL protocol only)
--sql	Start in SQL mode, auto-detecting protocol to use for connection
--sqlc	Start in SQL mode using a classic MySQL protocol connection
--ssl	Enable an SSL connection. Deprecated in version 1.0.10; use --ssl-mode instead
--ssl-ca	File that contains list of trusted SSL Certificate Authorities
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files
--ssl-cert	File that contains X.509 certificate
--ssl-cipher	Name of the SSL cipher to use
--ssl-crl	File that contains certificate revocation lists
--ssl-crlpath	Directory that contains certificate revocation list files
--ssl-key	File that contains X.509 key
--ssl-mode	Desired security state of connection to server
--table	Display output in table format
--tls-version	Permissible TLS protocol for encrypted connections
--uri	Session information in URI format
--user	MySQL user name to use when connecting to server (alias for --dbuser)
--version	Display version information and exit
--vertical	Display all SQL results vertically

- `--help`

  Display a help message and exit.

- `--auth-method=method`

  Authentication method to use for the account. Depends on the authentication plugin used for the account's password. MySQL Shell currently supports the following methods:

  - `mysql_native_password` - see Native Pluggable Authentication

  - `mysql_old_password` - see Old Native Pluggable Authentication

  - `sha256_password` - see Caching SHA-2 Pluggable Authentication

- `--classic`

  Creates a Classic session, to connect using MySQL Protocol.

- `--cluster`

  Ensures that the target server is part of an InnoDB cluster and if so, sets the `cluster` global variable to the cluster object.

- `--database=name`

  The default schema to use. This is an alias for `--schema`.

- `--dba=enableXProtocol`

  Enable X Plugin on connection with MySQL 5.7 server, so that you can use X Protocol connections for subsequent connections. Requires a connection using classic MySQL protocol.

  Not relevant for MySQL 8.0 servers, which have X Plugin enabled by default.

- `--dbpassword[=password]`, `-p[password]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the `password` value following the `--dbpassword` or `-p` option on the command line, you are prompted for one.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See End-User Guidelines for Password Security.

- `--dbuser=user_name`, `-u user_name`

  The MySQL user name to use when connecting to the server.

- `--execute=command`, `-e command`

  Execute the command using the currently active language and quit.

- `--file=file_name`, `-f file_name`

  Specify file to process in Batch mode.

- `--force`

  Continue processing in SQL and Batch modes even if errors occur.

- `--host=host_name`, `-h host_name`

  Connect to the MySQL server on the given host.

- `--get-server-public-key`

  MySQL Shell equivalent of `--get-server-public-key`.

  > **Important**
  >
  > Only supported with classic MySQL protocol connections.

  See Caching SHA-2 Pluggable Authentication.

- `--interactive[=full]`

  Emulate Interactive mode in Batch mode.

- `--js`

  Start in JavaScript mode.

- `--json[={pretty|raw}]`

  Print output in JSON format. With an option value of `pretty`, output is pretty-printed. With no option value or a value of `raw`, output is in raw JSON format.

- `--log-level=N`

  Specify the logging level. The value can be either an integer in the range from 1 to 8, or one of `none`, `internal`, `error`, `warning`, `info`, `debug`, `debug2`, or `debug3`. See Chapter 5, *MySQL Shell Application Log*.

- `-ma`

  Detects the session type automatically.

- `--mysql`

  Sets the session created at start up to create the connection using classic MySQL protocol.

- `--mysqlx`

  Sets the session created at start up to create the connection using X Protocol.

- `--node`

  Creates a Node session connected using X Protocol to a single server.

- `--name-cache`

  Enable automatic loading of table names based on the active default schema.

- `--no-name-cache`

  Disable loading of table names for autocompletion based on the active default schema and the DevAPI `db` object. Use `\rehash` to reload the name information manually.

- `--no-password`

  When connecting to the server, if the user has a passwordless account, which is insecure and not recommended, or if socket peer-credential authentication is in use (for Unix socket connections), you

must use `--no-password` to explicitly specify that no password is provided and the password prompt is not required.

- `--no-wizard`

  Disables the connection wizard which provides help when creating connections.

- `--passwords-from-stdin`

  Read the password from stdin.

- `--password[=password]`

  The password to use when connecting to the server. With a value provided, `--password` is an alias for `--dbpassword`. With an empty value provided, `--password=` has the same effect as `--no-password`, which specifies that no password is provided and the password prompt is not required.

- `--port=port_num`, `-P port_num`

  The TCP/IP port number to use for the connection. The default is port 33060.

- `--py`

  Start in Python mode.

- `--recreate-schema`

  Drop and recreate schema.

- `--redirect-primary`

  Ensures that the target server is part of an InnoDB cluster and if it is not a primary, finds the cluster's primary and connects to it. MySQL Shell exits with an error if any of the following is true when using this option:

  - Group Replication is not active

  - InnoDB cluster metadata does not exist

  - There is no quorum

- `--redirect-secondary`

  Ensures that the target server is part of an InnoDB cluster and if it is not a secondary, finds a secondary and connects to it. MySQL Shell exits with an error if any of the following is true when using this option:

  - Group Replication is not active

  - InnoDB cluster metadata does not exist

  - There is no quorum

  - The cluster is not single-primary and is running in multi-primary mode

  - There is no secondary in the cluster, for example because there is just one server instance

- `--schema=name`, `-D name`

  The default schema to use.

- `--server-public-key-path=file_name`

  MySQL Shell equivalent of `--server-public-key-path`.

  > **Important**
  >
  > Only supported with classic MySQL protocol connections.

  See `caching_sha2_password` plugin Caching SHA-2 Pluggable Authentication.

- `--show-warnings`

  Cause warnings to be shown after each statement if there are any.

- `--socket=path`, `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use. This option applies to Classic sessions only.

- `--sql`

  Start in SQL mode.

- `--sqlc`

  Start in SQL mode using a ClassicSession.

- `--sqln`

  Start in SQL mode using a NodeSession.

- `--sqlx`

  Start in SQL mode and create connection using X Protocol.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. The `mysqlsh` SSL options function in the same way as the SSL options for MySQL Server, see Command Options for Encrypted Connections for more information.

  `mysqlsh` accepts these SSL options: `--ssl-mode`, `--ssl-ca`, `--ssl-capath`, `--ssl-cert`, `--ssl-cipher`, `--ssl-crl`, `--ssl-crlpath`, `--ssl-key`. `--tls-version`.

- `--table`

  Display output in table format in Batch mode.

- `--uri=str`

  Create a connection upon startup, specifying the connection options in a URI string format, see Section 2.1.1, "Connecting using a URI String".

- `--user=user_name`

  The MySQL user name to use when connecting to the server. This is an alias for `--dbuser`.

- `--version`, `-V`

Display version information and exit.

- `--vertical`, `-E`

    Display results of SQL queries vertically.